Samuel Youssef
March 16, 2018

CSC 34200 - Computer Organization

Instructor: Prof. Zheng Peng

Lab **03**

- **Task 1:**

1. The entire source code for task 1 is:

```
                .data                           #data segment
nl:             .asciiz "\n"                    #ASCII for a new line
                .align 2                        #aligned at word boundary
course:         .asciiz "CSC343: "              #course name to be displayed
                .align 2                        #aligned at word boundary
name:           .asciiz "Samuel Youssef "       #name
                .align 2                        #aligned at word boundary
AndWord:        .asciiz "and "                  #"and" word to be displayed
                .align 2                        #aligned at word boundary
ID:             .asciiz "ID : 23402306"         #ID number to be dispalyed
                .align 2                        # aligned at word boudary
Fibword:        .asciiz "The Fibonacci Number F"  #fibonacci string to be displayed
                .align 2                        #aligned at word boundary
leftPar:        .asciiz "("                     #left parenthesis
                .align 2                        #aligned at word boundary
rightPar:       .asciiz ") "                    #right parenthesis
                .align 2                        #aligned at word boundary
isString:       .asciiz"is "                    #is word to be displayed
                .align 2                        #aligned at word boundary

                .text                           #code segment
                .globl main                     #declare main to be global
main:
                li $v0, 4                       #print string
                la $a0, course                  #load course name
                syscall                         #calling syscall to print string

                li $v0, 4                       #print string
                la $a0, name                    #loading name string to be printed
                syscall                         #calling syscall to print string

                li $v0, 4                       #print string
                la $a0, AndWord                 #loading AndWord string to be printed
                syscall                         # calling syscall to print string

                li $v0, 4                       #print string
                la $a0, ID                      #loading the ID string to be printed
                syscall                         #calling syscall to print string

                li $v0, 4                       #print string
                la $a0, nl                      #loading new line string to be printed
                syscall                         #calling syscall to print string


                li $t0, 0                       #intial value of F(n-2)
                li $t1, 1                       #intial value of F(n-1)

                addiu $s7, $0 , 1               #s7 holds the current index for the fib number to be calculated
                move $a1, $t0                   #moving the first argument in s1 to a1
                move $a2, $t1                   #move the second argument in s2 to a2
                jal print                       #call to function


loop:                                           #label of loop
                addiu $s7, $s7, 1               #adding 1 to current index
```

```
56              add $t2, $t0, $t1                   #adding the past two elements F(n-2)--> t0  and F(n-1)--> t1 and putting the result in t2
57              move $a1, $s7                       #moving the first argument in s1 to a1
58              move $a2, $t2                       #move the second argument in s2 to a2
59              jal print                           #calling function to print out the fib number
60              move $t0, $t1                       #reassinging F(n-2)
61              move $t1, $t2                       #reassinging F(n-1)
62              slti $t4, $s7, 10                   #set t4 to 1 if value in s7 is less than 10
63              beq  $t4, 1, loop                   #branch if value in t4 = 1 to loop
64
65
66
67              li $v0, 10                          #system call type 10, standard exit
68              syscall                             #call to OS
69
70
71  #list of registers used in print function
72  # 1) v0--> used to load system service numbers
73  # 2) a0--> used to load address of integer/string beign printed
74  # 3) s0--> used to hold the index of fib number
75  # 4) s1--> used to hold the fib number
76
77  # ------------------------ function print implementation--------------------------
78              .globl print                        #making the function print global to be accessed by other files
79  print:                                          #label function
80              li $v0, 4                           #system call to print string
81              la $a0, Fibword                     #loading the address of fibword to be printed
82              syscall                             #call to OS
83
84                  addiu $sp,$sp, -12                      #adjust the stack for 2 items(arguments of the function)
85                  sw $ra, 8($sp)                          #saving the return address on the stack
86                  sw $s0, 4($sp)                          #saving the content of the register s0 to be used inside the function
87                  sw $s1, 0($sp)                          #saving the content of register  s1 to be used in the function
88
89                  move $s0, $a1                           #moving the content of a1 (index of the element) to s0
90                  move $s1, $a2                           #moving the content of a2 (element) to s1
91
92                  li $v0, 4                               #system call to print string
93                  la $a0,leftPar                          #loading the address of leftPar to be printed
94                  syscall                                 #call to OS
95
96                  li $v0, 1                               #system call to print integer
97                  la $a0, ($s0)                           #loading address of index of fib number
98                  syscall                                 #call to OS
99
100                 li $v0, 4                               #system call to print string
101                 la $a0, rightPar                        #loading the address of rightPar
102                 syscall                                 #call to OS
103
104                 li $v0, 4                               #system call to print string
105                 la $a0, isString                        #loading address of isString
106                 syscall                                 #call to OS
107
108                 li $v0, 1                               #system call to print integer
109                 la $a0, ($s1)                           #loading address of fib number
110                    syscall                                      #call to OS
111
112                    li $v0, 4                                    #system call to print string
113                    la $a0, nl                                   #loading address of new line
114                    syscall                                      #call to OS
115
116                    lw $s1, 0($sp)                               #restoring the content of the register s1
117                    lw $s0, 4($sp)                               #restoring the content of the register s0
118                    lw $ra, 8($sp)                               #restoring the content of the register ra
119                    addiu $sp, $sp, 12                           #restoring stack pointer
120
121                    jr $ra                                       #return statement
122
```

figure 1: the entire source code for task 1.

2. The program outputs are as follows:

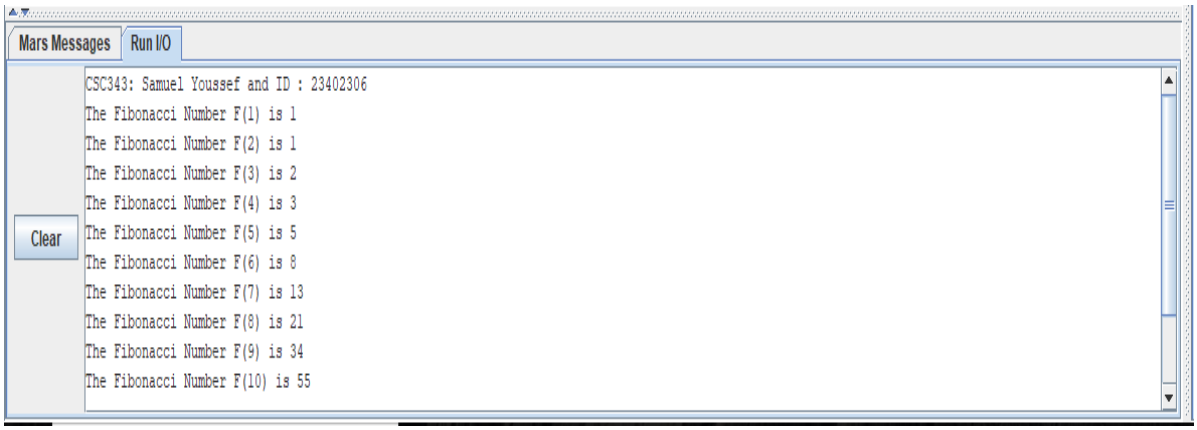


Figure 2: output of task 1.

3. The LA pseudo instruction consists of two instructions:
   a. The LUI (load upper immediate) instruction in figure 3.
   b. The ORI (ors a register value with and immediate value) in figure 4.

LUI -- ***Load upper immediate***

Description:	The immediate value is shifted left 16 bits and stored in the register. The lower 16 bits are zeroes.
Operation:	$t = (imm << 16); advance_pc (4);
Syntax:	lui $t, imm
Encoding:	0011 11-- ---t tttt iiii iiii iiii iiii

Figure 3: LUI instruction.

ORI -- ***Bitwise or immediate***

Description:	Bitwise ors a register and an immediate value and stores the result in a register
Operation:	$t = $s | imm; advance_pc (4);
Syntax:	ori $t, $s, imm
Encoding:	0011 01ss ssst tttt iiii iiii iiii iiii

Figure 4: ORI instruction.

→Explanation of LA instruction:

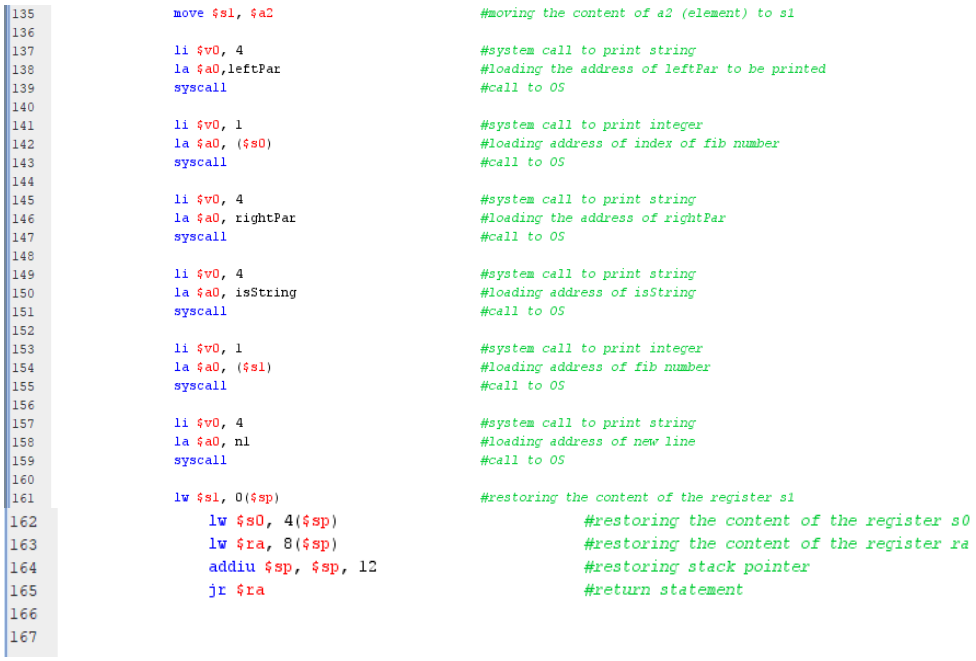
- When the LA (pseudo instruction) is used, the assembler converts this instruction into 2 instructions to be executed. The assembler does so because it cannot specify a full 32-bit immediate address value. The first sub instruction is LUI instruction which loads the value of $1

(also named $at (a register reserved for pseudo instructions)) with the base address of the data segment. After that the base address in $at is shifted left by 16 bits; that means that the lower 16 bits of $at are all set to zero. And the upper 16 bits of $at contains the base address. The second sub instruction is the ORI instruction. This instruction ors the value of $at (upper half is the base address and lower half is all zeroes) with another immediate value that contains the offset address of the targeted piece of data in the data segment. After bitwise OR process ends, the destination register in the ORI instruction now contains the full address of the that piece of data.

- **Task 2:**
  1. The entire source code of task 2 is:

```
                .data                           #data segment
n1:             .asciiz "\n"                    #ASCII for a new line
                .align 2                        #aligned at word boundary
course:         .asciiz "CSC343: "              #course name to be displayed
                .align 2                        #aligned at word boundary
name:           .asciiz "Samuel Youssef "       #name
                .align 2                        #aligned at word boundary
AndWord:        .asciiz "and "                  #"and" word to be displayed
                .align 2                        #aligned at word boundary
ID:             .asciiz "ID : 23402306"         #ID number to be dispalyed
                .align 2                        # aligned at word boudary
Fibword:        .asciiz "The Fibonacci Number F"#fibonacci string to be displayed
                .align 2                        #aligned at word boundary
leftPar:        .asciiz "("                     #left parenthesis
                .align 2                        #aligned at word boundary
rightPar:       .asciiz ") "                    #right parenthesis
                .align 2                        #aligned at word boundary
isString:       .asciiz"is "                    #is word to be displayed
                .align 2                        #aligned at word boundary
FBN:            .space 400                      #array to hold fib numbers
                .align 2                        #aligned at word boundary
NUM_FBN:        .space 4                        #allocating space to hold the index number
                .align 2                        #aligned at word boundary

                .text                           #code segment
                .globl main                     #declare main to be global
main:

                move $s0, $zero                  #s0 holds f[0] = 0
                addiu $s1, $zero, 1              #s1 hold f[1] = 1
                la $t0, FBN                      #t0 holds the address of the Fib array
                move $t1, $zero             #t1 to increament index value
                addu $t2, $t1, $t0               #t2 to hold current array element being assigned
                sw $s0, 0($t2)                   #saving s0 = 0 into the first in index 0 of array fib
                addiu $t1, $t1, 4                #increasing the index
                addu $t2, $t1, $t0               #getting access to the appropriate index for writitng
                sw $s1, 0($t2)                   #saving s1 = 1 into index 1 of array fib
                la $s3, NUM_FBN                  #hold the address of num_fbn


                li $v0, 4                        #print string
                la $a0, course                   #load course name
                syscall                          #calling syscall to print string

                li $v0, 4                        #print string
                la $a0, name                     #loading name string to be printed
                syscall                          #calling syscall to print string

                li $v0, 4                        #print string
                la $a0, AndWord                  #loading AndWord string to be printed
                syscall                          # calling syscall to print string

                li $v0, 4                        #print string
                la $a0, ID                       #loading the ID string to be printed
                syscall                          #calling syscall to print string
```

```
                li $v0, 4                               #print string
                la $a0, nl                              #loading new line string to be printed
                syscall                                 #calling syscall to print string

                addiu $t8, $zero, 4                     # N = 1
                move $t3, $zero                         #holds element F[n-1]
                move $t4, $zero                         #holds element F[n]
                move $t7, $zero                         #holds the address of f[n-1]
                move $t9, $zero                         #holds the address of f[n]
                move $s2, $zero                         #hold the boolean value to check if a certain register is greater or less than zero

loop:                                                   #label of loop
                move $t5, $t8                           #holds the value of N (tracking index f[n-1])
                move $t6, $t8                           #holds the value of N (tracking index f[n])
                addi $t5, $t5, -4                       #doing N-1
                add $t7, $t0, $t5                       #getting the address of f[n-1]
                add $t9, $t0, $t6                       #getting the address of f[n]
                lw $t3, 0($t7)                          #getting the element f[n-1]
                lw $t4, 0($t9)                          #getting the element f[n]
                addu $t4, $t4, $t3                      #doing unsigned addition on f[n-1] and f[n]
                slt  $s2, $t4, $zero                    #checking if the addition is overflow or not
                bne $s2, $zero, continuetoPrint         #end of operation (overflow exits) going to print loop
                addi $t8, $t8, 4                        #keeping track of the index of the element
                sw $t4, 4($t9)                          #assigning appropriate element of the array
                j loop                                  #looping until overflow happens


continuetoPrint:
                sw $t8, 0($s3)                          #assigning num_fbn to the number of time computation loop happened
                move $t8, $zero                         #writing a new value to t8 = 0.
                move $s4, $zero                         #holds the address of the element being printed out
                li  $s5, 4                              #holds a constant number 4
                move $s6, $zero                         #holds content of num_fbn

loopPrint:
                addu $s4, $t0, $t8                      #having the correct current address of element printing out saved into s4
                div $t8, $s5                            #having the index number calculated
                mflo $a1                                #moving the index from the lo register to a1
                lw $a2, 0($s4)                          #loading the element in a2
                jal print                               #calling the function of print
                addi $t8, $t8, 4                        #adding 4 bytes for the index of the next element
                lw $s6, 0($s3)                          #loading the number of elements we have computed
                slt $s2, $t8, $s6                       #check if we exceeded the number of elements we have originally calculated by comparing to t8
                beq $s2, $zero, terminate               #if t8 is greater than the number of elements that we have calculated, we terminate the program
                j loopPrint                             #reiterate the loop if condition in last statement is not satisfied


terminate:                                              # printing the last element of the array

                addu $s4, $t0, $t8                      #having the correct current address of element printing out saved into s4
                div $t8, $s5                            #having the index number calculated
                mflo $a1                                #moving the index from the lo register to a1
                lw $a2, 0($s4)                          #loading the element in a2
                 lw $a2, 0($s4)                             #loading the element in a2
                 jal print                                  #calling the function of print

                 li $v0, 10                                 #system call type 10, standard exit
                 syscall                                    #call to OS


#list of registers used in print function
# 1) v0--> used to load system service numbers
# 2) a0--> used to load address of integer/string beign printed
# 3) s0--> used to hold the index of fib number
# 4) s1--> used to hold the fib number

# ------------------------- function print implementation---------------------------
                .globl print                            #making the function print global to be accessed by other files
print:                                                  #label function
                li $v0, 4                               #system call to print string
                la $a0, Fibword                         #loading the address of fibword to be printed
                syscall                                 #call to OS

                addiu $sp,$sp, -12                      #adjust the stack for 2 items(arguments of the function)
                sw $ra, 8($sp)                          #saving the return address on the stack
                sw $s0, 4($sp)                          #saving the content of the register s0 to be used inside the function
                sw $s1, 0($sp)                          #saving the content of register  s1 to be used in the function

                move $s0, $a1                           #moving the content of a1 (index of the element) to s0
```



```
135             move $s1, $a2                   #moving the content of a2 (element) to s1
136
137             li $v0, 4                       #system call to print string
138             la $a0,leftPar                  #loading the address of leftPar to be printed
139             syscall                         #call to OS
140
141             li $v0, 1                       #system call to print integer
142             la $a0, ($s0)                   #loading address of index of fib number
143             syscall                         #call to OS
144
145             li $v0, 4                       #system call to print string
146             la $a0, rightPar                #loading the address of rightPar
147             syscall                         #call to OS
148
149             li $v0, 4                       #system call to print string
150             la $a0, isString                #loading address of isString
151             syscall                         #call to OS
152
153             li $v0, 1                       #system call to print integer
154             la $a0, ($s1)                   #loading address of fib number
155             syscall                         #call to OS
156
157             li $v0, 4                       #system call to print string
158             la $a0, nl                      #loading address of new line
159             syscall                         #call to OS
160
161             lw $s1, 0($sp)                  #restoring the content of the register s1
162                 lw $s0, 4($sp)                      #restoring the content of the register s0
163                 lw $ra, 8($sp)                      #restoring the content of the register ra
164                 addiu $sp, $sp, 12                  #restoring stack pointer
165                 jr $ra                              #return statement
166
167
```

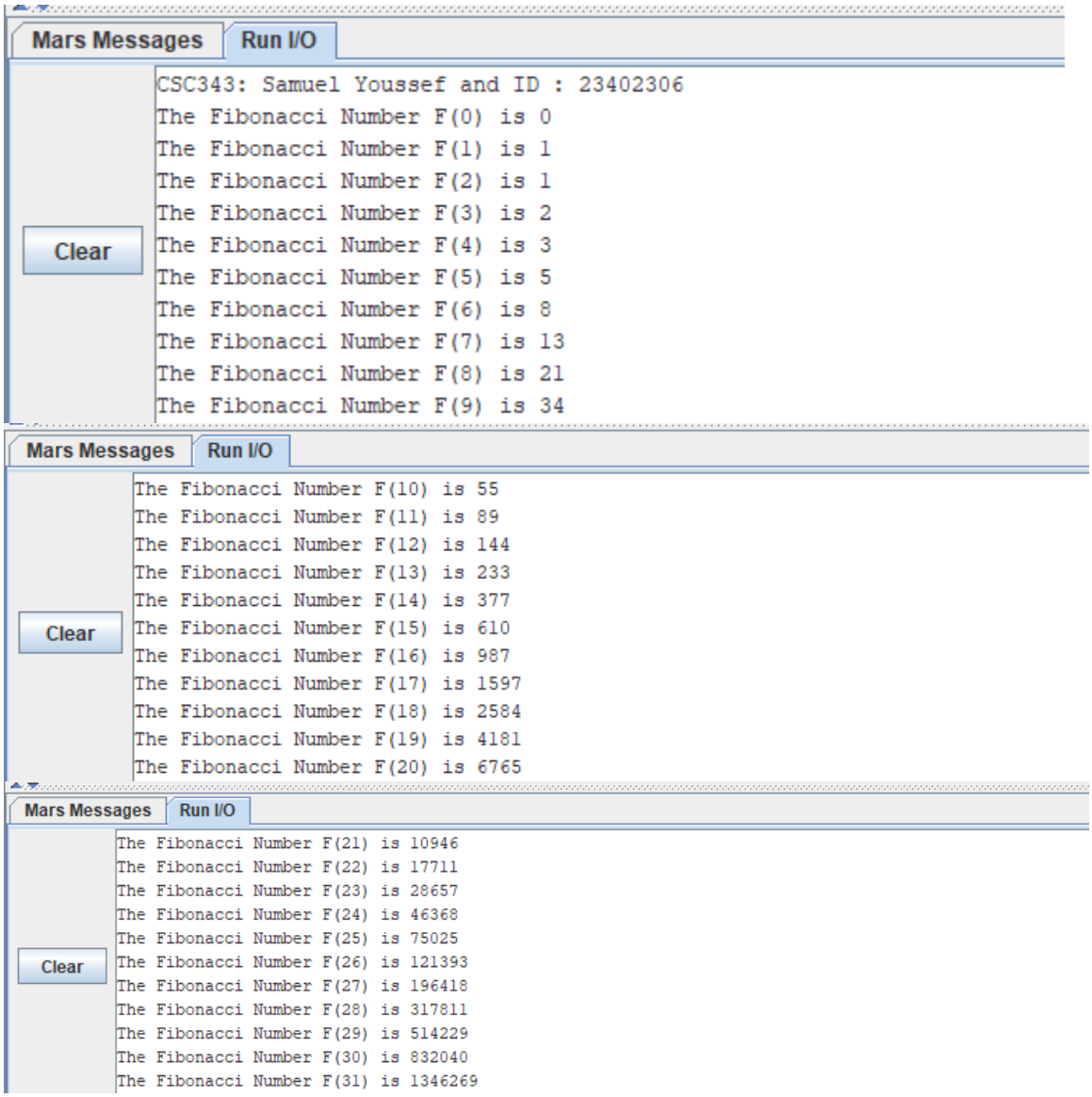
Figure 5: the entire source code of task 2.

2. In this lab, we used the unsigned instructions of addition (addu, and addiu) and we assumed that all numbers are signed. The biggest 32-bit signed binary number is $2^{31} - 1 = 2{,}147{,}483{,}647$ ($7FFF, FFFF_{16}$). We checked if there is an overflow by using ***slt*** instruction as illustrated in figure 6. If the value of addition exceeds the number ($2^{31} - 1 = 2{,}147{,}483{,}647$), then there is an overflow and the addition of two positive integers generates a negative value. That is the reason why we checked to see if the destination register of the addition process contained a value less than zero or not. Task 2 generated Fibonacci numbers (from index 0 to index 46). Fibonacci number (47^{th}) is 2,971,215,073 which is greater than the biggest signed value a 32-bit register can hold. So obviously the loop terminated after calculating the 47^{th} Fibonacci value ***without*** storing that value in the FBN array.

```
76              addu $t4, $t4, $t3              #doing unsigned addition on f[n-1] and f[n]
77              slt  $s2, $t4, $zero            #checking if the addition is overflow or not
```

Figure 6: Statements in Task 2 to check if the addition process result in overflow.

3. The program (of task 2) output is as follows:



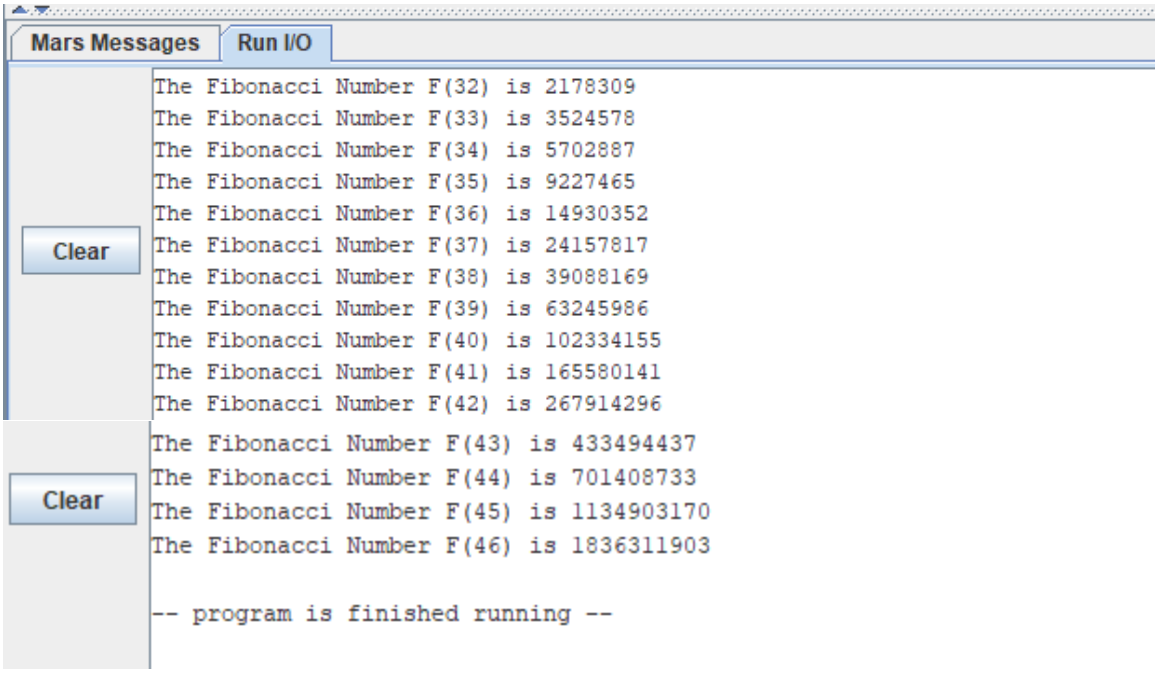


Figure 7: output of task 2.